Hyperparameter Tuning is All You Need for LISTA

Anonymous Author(s)
Affiliation
Address
email

Abstract

Learned Iterative Shrinkage-Thresholding Algorithm (LISTA) introduces the concept of unfolding an iterative algorithm and trains it like a neural network. It had great success on sparse recovery. In this paper, we show that adding momentum to the LISTA network achieves a better convergence rate and, in particular, the network with instance-optimal parameters is superlinearly convergent. Moreover, our new theoretical results lead to a practical approach of automatically and adaptively calculating the parameters of a LISTA network layer based on its previous layers. Perhaps most surprisingly, such an adaptive-parameter procedure reduces the training of LISTA to tuning **only three hyperparameters** from data: a new record set in the context of the recent advances on trimming down LISTA complexity. We call this new ultra-light weight network *HyperLISTA*. Compared to state-of-the-art LISTA models, HyperLISTA achieves almost the same performance on seen data distributions and performs better when tested on unseen distributions (specifically, those with different sparsity levels and nonzero magnitudes). We will release codes.

1 Introduction

In this paper, we study the sparse linear inverse problem, where we strive to recover an unknown sparse vector $\boldsymbol{x}^* \in \mathbb{R}^n$ from its noisy linear measurement $\boldsymbol{b}$ generated from

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}^* + \varepsilon, \tag{1}$$

where $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is the *dictionary*, and $\varepsilon \in \mathbb{R}^m$ is additive Gaussian white noise. For simplicity, each column of $\boldsymbol{A}$, is normalized. Typically, we have much fewer rows than columns in the dictionary $\boldsymbol{A}$, i.e., $m \ll n$. Therefore, Equation (1) is an under-determined system. The sparse inverse problem, also known as sparse coding, plays essential roles in a wide range of applications including feature selection, signal reconstruction and pattern recognition.

Sparse linear inverse problems are well studied in the literature of optimization. For example, it can be approximated by LASSO [24] and solved by many optimization algorithms [9, 4]. These *model-based* solutions explicitly consider and incorporate the sparsity prior and usually exploit iterative routines. Deep-learning-based approaches are proposed for empirically solving inverse problems recently [19], which produce black-box models that are trained with data in an end-to-end way, while somehow ignoring the sparsity prior. We call these *model-free* methods. Comparing the two streams, model-based methods take hundreds to thousands of iterations to generate accurate solutions, while model-free methods, if properly trained, can achieve similar accuracy in tens of layers. However, model-based methods come with data-agnostic convergence (rate) guarantees under suitable conditions, whereas model-free methods only empirically work for instances similar to the training data, lacking theoretical guarantees and interpretability. More discussions are found in [6].

Unrolling is an uprising approach to the sparse recovery problems that bridges the two streams [16]. By converting each iteration of a classic iterative algorithm into one layer of a neural network, one can unroll and truncate a model-based method into a feed-forward neural network, with a

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

finite number of layers. Relevant parameters (e.g., the dictionary, step sizes and thresholds) in the original algorithm are transformed into learnable weights and are trained on data. [12] pioneered the unrolling scheme for for solving sparse coding and achieved great empirical success. By unrolling the Iterative Shrinkage-Thresholding Algorithm (ISTA), the authors demonstrated up to two magnitudes of acceleration of the learned ISTA (LISTA) compared to the original ISTA. The similar unrolling idea was later extended to numerous optimization problems and algorithms [23, 21, 11, 29, 2, 20, 13, 8].

The balance between empirical performance and interpretability of unrolling motivates theoretical efforts to explain their acceleration success. [17, 27] explains the acceleration of unrolled ISTA from different perspectives, e.g., matrix factorization and the improved restricted isometry property. [7] proved for the first time the theoretical worst-case linear convergence of unrolled ISTA models under mild conditions with proper selections of the learnable weights. [14] further found that the optimal selection of most learnable weights can be analytically inferred from the dictionary and reduced the parameter space of unrolled ISTA to only a series of scalars. However, such convergence guarantees only hold on data that are drawn from exactly the same distribution of the training samples (for example, with the same sparsity level in sparse vectors and fixed distributions for the magnitudes of non-zero elements and additive noises). When a sample falls out of that distribution, the trained unrolled model will see significantly performance degradation or even divergence.

1.1 Related Works

The balance between empirical performance and interpretability of unrolling motivates a line of theoretical efforts to explain their acceleration success. [17] attempted to understand LISTA by re-factorizing the Gram matrix of the dictionary $\boldsymbol{D}$ in (1), trying to diagonalize the Gram matrix under a basis that produces small perturbations in ℓ_1 balls. The authors proved that such re-parameterization converges sublinearly and empirically showed that it achieved similar acceleration gain to LISTA. [27] investigated the variant of unrolling iterative hard thresholding (IHT) and proved the existence of a matrix transformation that can improve the restricted isometry property constant of the original dictionary. Although such transformation is difficult to optimize, the authors argued that the data-driven training process practically learned that, and hence the acceleration effect. They also proved that a form of weight coupling is necessary for the ground truth sparse vectors to be fixed points.

A seminal work [7] extended the weight coupling necessary condition in [27] to LISTA. The authors for the first time proved the existence of a learnable parameter set that guarantees linear convergence of LISTA . They also introduced the support selection technique that provides significant acceleration to LISTA. Later in [14], they further showed that the weight matrix in [7] can be analytically derived from the dictionary $\boldsymbol{D}$ and free from learning, reducing the learnable parameter set to only tens of scalars (layer-wise step sizes and thresholds). The resulting algorithm, called Analytic LISTA (ALISTA), represents a milestone in simplifying LISTA. [25] extends ALISTA by introducing gain and overshoot gating mechanisms and integrate them into unrolling.

Further, the theories in [14] have revealed the inherent principle of the “optimal" model weights, by select thresholding parameters using the step size and the current recovery error. However, that rule can not be practically used during inference due to inaccessibility of ground truth sparse codes. [30] proposed a workaround solution called error-based thresholding (EBT), which leverages a function of the layer-wise reconstruction error to suggest an appropriate threshold value for each observation on each layer. Their approach also reduces the LISTA parameter space to one learnable scalar per layer, and making the learned LISTA more adaptive to various observations. Another interesting work [5] uses a black-box LSTM model that takes the historical residual and update quantity as input and generates layerwise step sizes and thresholds in ALISTA. Their proposed model, called Neurally Augmented ALISTA (NA-ALISTA), performed well in large-scale problems.

1.2 Our Contributions

Our research question is along the line of [14, 30, 5]: *can we improve the LISTA parameterization, by further disentangling the the learnable parameters from the model-based observable terms (e.g., reconstruction errors, intermediate outputs)*? Our aim is to create more light-weight, albeit more robust LISTA models with “baked in" adaptivity to testing samples from unseen distributions. The goal should be achieved without sacrificing our previously attained benefits: empirical performance, and convergence (rate) guarantees – if not improving them further.

We present an affirmative answer to the above question by learning instance-optimal parameters with a new ALISTA parameterization. First, we start with a new symmetric parameterization for the weight matrix in ALISTA so that ALISTA also converges to LASSO function minimizers, that lays the foundation of momentum-based analysis. Second, we prove that by augmenting ALISTA with momentum, the new unrolled network can achieve a better convergence rate over ALISTA. Third, we find that once the above LISTA network is further enabled with instance-optimal parameters, then a superlinear convergence can be attained. Lastly, we show that those instance-optimal parameters of each layer can be generated in closed forms, consequently reducing ALISTA to only learning **three instance- and layer-invariant hyperparameters**.

Our new ultra-light weight network is termed as *HyperLISTA*. Compared to ALISTA, the new parameterization of HyperLISTA facilitates new backpropagation-free "training" options (e.g., bayesian optimization), and can unroll to more iterations in testing than training thanks to its layer-wise decomposed form. When comparing to state-of-the-art LISTA variants in experiments, HyperLISTA achieves the same strong performance on seen data distributions, and performs better when tested on unseen distributions (e.g., with different sparsity levels and nonzero magnitudes). In the supplementary materials, we also show the superlinear convergence can be observed in HyperLISTA experiments, when our automatic parameter tuning is performed per instance.

2 Assumptions and Preliminaries

Setup and Notations Throughout this paper, we refer "parameters" as learnable weights, thresholds $\theta^{(k)}$, step sizes $\gamma^{(k)}$ that are directed defined in LISTA models and usually trained by back-propagation. In contrast, we refer "hyperparameters" as those constants that are often ad-hoc pre-defined to calculate parameters. For example, $\theta^{(k)} = c_1 \|\boldsymbol{A}^+(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})\|_1$, where $\theta^{(k)}$ is a parameter and c_1 is a hyperparameter. We represent a matrix as a capital bold letter $\boldsymbol{W}$, $\mathrm{diag}(\boldsymbol{W})$ means the diagonal elements of matrix $\boldsymbol{W}$ and the pseudoinverse of a matrix is denoted by $\boldsymbol{W}^+$. A vector is represented as a lowercase bold letter $\boldsymbol{x}$ and its entries are represented as lowercase regular letters with lower indices $\boldsymbol{x} = [x_1, x_2, \cdots, x_3]^T$. Upper indices represents iterations or layers. Moreover, $\mathbf{0}, \mathbf{1}$ represent a vector with all zeros or all ones, respectively. "$\mathrm{supp}(\boldsymbol{x})$" denotes the support, the indices of all nonzeros in vector $\boldsymbol{x}$.

With an ℓ_1 penalty term, LASSO $\min_{\boldsymbol{x}}(1/2)\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1$ provides an estimate of the solution of the sparse recovery problem (1). A popular algorithm to solve LASSO is Iterative Shrinkage-Thresholding Algorithm (ISTA):

$$\boldsymbol{x}^{(k+1)} = \eta_{\lambda/L}\Big(\boldsymbol{x}^{(k)} + \frac{1}{L}\boldsymbol{A}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)})\Big), \quad k = 0, 1, 2, \ldots$$

where η_θ is the soft-thresholding function: $\eta_\theta(\boldsymbol{x}) = \mathrm{sign}(\boldsymbol{x})\max(0, |\boldsymbol{x}| - \theta)$ and L is usually taken as the largest eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$.

The original LISTA work [12] parameterizes ISTA as

$$\boldsymbol{x}^{(k+1)} = \eta_{\theta^{(k)}}\Big(\boldsymbol{W}_1^{(k)}\boldsymbol{x}^{(k)} + \boldsymbol{W}_2^{(k)}\boldsymbol{b}\Big), \quad k = 1, 2, 3, \ldots, \tag{2}$$

where $\Theta = \{\boldsymbol{W}_1^{(k)}, \boldsymbol{W}_2^{(k)}, \theta^{(k)}\}$ are learnable parameters to train from data. In the training process, (2) are unfolded and truncated to K steps and the parameters are trained by minimizing the following loss function defined over the distribution of training samples D_{tr}:

$$\underset{\Theta}{\mathrm{minimize}}\, \mathbb{E}_{(\boldsymbol{x}^*, \boldsymbol{b}) \sim D_{\mathrm{tr}}} \Big\| \boldsymbol{x}^{(K)}(\Theta, \boldsymbol{b}, \boldsymbol{x}^{(0)}) - \boldsymbol{x}^* \Big\|_2^2. \tag{3}$$

ALISTA [14] advances LISTA with a lighter and more interpretable parameterization , where the learnable parameters are $\Theta = \{\gamma^{(k)}, \theta^{(k)}\}$ and the iterations are formulated as

$$\boldsymbol{x}^{(k+1)} = \eta_{\theta^{(k)}}^{p^{(k)}}\Big(\boldsymbol{x}^{(k)} + \gamma^{(k)}\boldsymbol{W}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)})\Big), \tag{4}$$

where $\boldsymbol{W}$ is calculated by solving the following optimization problem:

$$\boldsymbol{W} \in \underset{\boldsymbol{W} \in \mathbb{R}^{m \times n}}{\mathrm{argmin}} \left\| \boldsymbol{W}^T\boldsymbol{A} - \boldsymbol{I} \right\|_F^2, \quad \text{s.t. } \mathrm{diag}(\boldsymbol{W}^T\boldsymbol{A}) = \mathbf{1}. \tag{5}$$

The thresholding operator is improved with support selection. At layer k, a certain percentage of entries with largest magnitudes are trusted as "true support" and won't pass them through thresholding. Specifically, the i-th element of $\eta_{\theta^{(k)}}^{p^{(k)}}(\boldsymbol{v})$ is defined as

$$(\eta_{\theta^{(k)}}^{p^{(k)}}(\boldsymbol{v}))_i = \begin{cases} v_i & : v_i > \theta^{(k)}, \quad i \in S^{p^{(k)}}(\boldsymbol{v}), \\ v_i - \theta^{(k)} & : v_i > \theta^{(k)}, \quad i \notin S^{p^{(k)}}(\boldsymbol{v}), \\ 0 & : -\theta^{(k)} \leq v_i \leq \theta^{(k)} \\ v_i + \theta^{(k)} & : v_i < -\theta^{(k)}, \quad i \notin S^{p^{(k)}}(\boldsymbol{v}), \\ v_i & : v_i < -\theta^{(k)}, \quad i \in S^{p^{(k)}}(\boldsymbol{v}), \end{cases} \tag{6}$$

where $S^{p^{(k)}}(\boldsymbol{v})$ includes the elements with the largest $p^{(k)}$ magnitudes in vector $v \in \mathbb{R}^n$:

$$S^{p^{(k)}}(\boldsymbol{v}) = \left\{ i_1, i_2, \cdots, i_{p^{(k)}} \middle| |v_{i_1}| \geq |v_{i_2}| \geq \cdots |v_{i_{p^{(k)}}}| \cdots \geq |v_{i_n}| \right\}. \tag{7}$$

With $p^{(k)} = 0$, the operator reduces to the soft-thresholding and, with $p^{(k)} = n$, the operator reduces to the hard-thresholding. Thus, operator $\eta_{\theta^{(k)}}^{p^{(k)}}(\boldsymbol{v})$ is actually a balance between soft- and hard-thresholding. As k small, $p^{(k)}$ is usually chosen as a small fraction of n, the operator tends to not trust the signal, and vice versa. In ALISTA, $p^{(k)}$ is regarded as a hyperparameter to be manually tuned by cross validation or so. Later in this work, we will present it as a parameter that could be calculated by the information from previous iterations (layers).

Assumption 1 (Basic assumptions)**.** *Signal $\boldsymbol{x}^*$ is sampled from the following set:*

$$\boldsymbol{x}^* \in \mathcal{X}(B, \underline{B}, s) \triangleq \left\{ \boldsymbol{x}^* \middle| 0 < \underline{B} \leq |\boldsymbol{x}_i^*| \leq B, \forall i, \ \|\boldsymbol{x}^*\|_0 \leq s \right\}. \tag{8}$$

In other words, $\boldsymbol{x}^*$ is bounded and sparse. Furthermore, we assume $\varepsilon = 0$.

Compared with the Assumption 2 in [7], the assumption of "$|x_i^*| > \underline{B} > 0$" is slightly stronger. But we will justify (after Theorem 1) that uniformly under Assumption 1 in this paper, our method also achieves a better convergence rate than state-of-the-arts. Moreover, we assume zero-noise for simplicity of the proofs but our experiments will show that our model works well on noisy cases.

3 Methodologies and Theories

3.1 Preparation: Symmetric Jacobian of Gradients

Formula (5) provides good weights that have low coherence with matrix $\boldsymbol{A}$ [14]. However, the matrix $\boldsymbol{W}^T\boldsymbol{A}$ is not symmetric and the vector $\boldsymbol{W}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})$ is not a gradient of a function. Consequently, ALISTA cannot converge to the minimizer of $F(\boldsymbol{x}) = (1/2)\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1$ and one can only train the model given the ground truth $\boldsymbol{x}^*$ by loss functions like $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_2^2$ [3]. In this paper, we adopt another method to calculate $\boldsymbol{W}$. Inspired by [1], we define $\boldsymbol{W} = (\boldsymbol{G}^T\boldsymbol{G})\boldsymbol{A}$ (the matrix $\boldsymbol{G} \in \Re^{m\times m}$ is named as the Gram matrix), and get the following problem instead of (5):

$$\min_{\boldsymbol{G}} \|\boldsymbol{A}^T\boldsymbol{G}^T\boldsymbol{G}\boldsymbol{A} - \boldsymbol{I}\|_F^2, \quad \text{s.t. } \operatorname{diag}(\boldsymbol{A}^T\boldsymbol{G}^T\boldsymbol{G}\boldsymbol{A}) = \boldsymbol{1}. \tag{9}$$

However, the constraint in the above problem (9) is hard to handle. Thus, we introduce an extra matrix $\boldsymbol{D} = \boldsymbol{G}\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and use the following method instead to calculate dictionary:

$$\min_{\boldsymbol{G},\boldsymbol{D}} \|\boldsymbol{D}^T\boldsymbol{D} - \boldsymbol{I}\|_F^2 + \frac{1}{\alpha}\|\boldsymbol{D} - \boldsymbol{G}\boldsymbol{A}\|_F^2, \quad \text{s.t. } \operatorname{diag}(\boldsymbol{D}^T\boldsymbol{D}) = \boldsymbol{1}. \tag{10}$$

With proper $\alpha > 0$, the solution of (10) approximates (9) well. We use an adaptive rule to choose α and adpot a variant of the algorithm (described in details in the supplement) proposed in [15] to solve (10). Such formula leads to a symmetric matrix $\boldsymbol{W}^T\boldsymbol{A} = (\boldsymbol{G}\boldsymbol{A})^T(\boldsymbol{G}\boldsymbol{A})$. With this symmetry, the vector $\boldsymbol{W}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})$ is actually the gradient of function $(1/2)\|\boldsymbol{G}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\|_2^2$. One may train model (4) using methods in [3] with the following loss function without knowing the ground truth:

$$F(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{G}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\|_2^2 + \lambda\|\boldsymbol{x}\|_1. \tag{11}$$

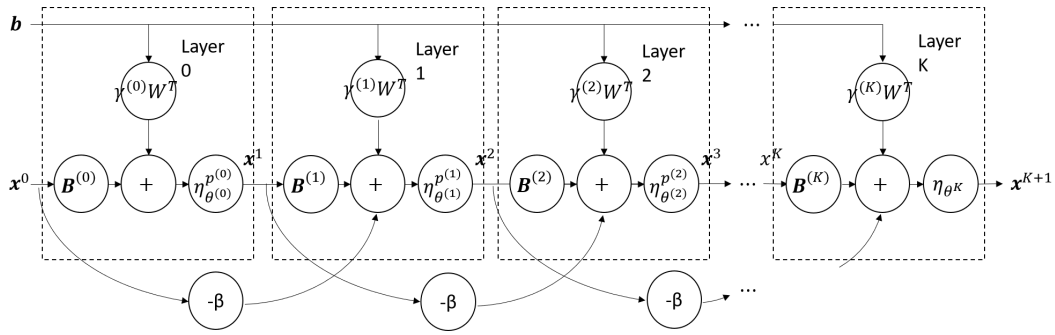


Figure 1: ALISTA with momentum: $\boldsymbol{B}^{(0)} = \boldsymbol{I} - \gamma^{(0)}\boldsymbol{W}^T\boldsymbol{A}, \boldsymbol{B}^{(k)} = \big(1+\beta^{(k)}\big)\boldsymbol{I} - \gamma^{(k)}\boldsymbol{W}^T\boldsymbol{A}$ $(k \geq 1)$.

On the other hand, it usually holds in practice that

$$\|\boldsymbol{W}^T\boldsymbol{A} - \boldsymbol{I}\|_F \approx \|(\boldsymbol{G}\boldsymbol{A})^T(\boldsymbol{G}\boldsymbol{A}) - \boldsymbol{I}\|_F, \tag{12}$$

where $\boldsymbol{W}$ is calculated by (5) and $\boldsymbol{G}$ is calculated by (10). We validate (12) by numerical results in Section 4.1. Such conclusion demonstrates that procedures (10) and (5) lead to similar mutual coherence and, consequently, one can also pursue $\boldsymbol{x}^{(k)} \to \boldsymbol{x}^*$ with the new dictionary $\boldsymbol{G}\boldsymbol{A}$, with almost no performance loss.

3.2 ALISTA with Momentum: Improved Linear Convergence Rate

Besides letting ALISTA be able to converge to LASSO minimizers, the above new symmetric parameterization make us be able to unify the proof schemes in ALISTA and Polyak's heavy ball momentum, which was not feasible previously.

As is well known in the optimization community, adding a momentum allows the iterative algorithm to move faster along dimensions of low curvature where the update is consistently small but in the same direction. Two classical momentum algorithms were respectively proposed by Polyak [22] and Nesterov [18]. Nesterov's algorithm alternates between the gradient update and the extrapolation, while Polyak's algorithm can be written as an iterative algorithm within the space of gradient updates by simply adding a momentum term. To avoid extra training overhead, we use Polyak's heavy ball scheme and add a momentum term to the ALISTA formula for $k \geq 1$[1]:

$$\boldsymbol{x}^{(k+1)} = \eta_{\theta^{(k)}}^{p^{(k)}}\Big(\boldsymbol{x}^{(k)} + \gamma^{(k)}\boldsymbol{W}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k)}) + \beta^{(k)}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)})\Big) \tag{13}$$

Note there are other LISTA variants that incorporated additional momentum forms [17, 26], but they did not contribute to provably accelerated convergence rate.

In the context of (A)LISTA, model (13) can provide a faster linear convergence rate than that of ALISTA (with the same settings). Defining constant μ as the mutual coherence of matrix $\boldsymbol{D}$ (defined in (10)): $\mu := \max_{i\neq j}(\boldsymbol{D}_{:,i})^T\boldsymbol{D}_{:,j}$, we have the following theorem.

Theorem 1 (Convergence of ALISTA-Momentum)**.** *Let $\boldsymbol{x}^{(0)} = \boldsymbol{0}$ and $\{\boldsymbol{x}^{(k)}\}_{k=1}^{\infty}$ be generated by (13). If Assumption 1 holds and $s < (1+1/\mu)/2$, then there exists a sequence of parameters $\{\theta^{(k)}, p^{(k)}, \gamma^{(k)}, \beta^{(k)}\}_{k=1}^{\infty}$ such that, for all $\boldsymbol{x}^* \in \mathcal{X}(B, \underline{B}, s)$, we have the error bound:*

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_2 \leq \sqrt{s}B\prod_{t=0}^{k} c^{(t)}, \quad \forall k = 1, 2, \cdots, \tag{14}$$

where $c^{(k)} > 0$ satisfies:

$$c^{(k)} \leq 2\mu s - \mu < 1, \quad \forall k \tag{15}$$

$$c^{(k)} \leq 1 - \sqrt{1 - \mu s + \mu} < 2\mu s - \mu, \quad \forall k \geq \frac{\log(\underline{B}) - \log(sB)}{\log(2\mu s - \mu)} \tag{16}$$

[1] As $k = 0$, we keep the ALISTA formula since $\boldsymbol{x}^{(-1)}$ is not defined.

and the parameter sequence satisfies

$$\begin{aligned} \theta^{(k)} =& \mu \sup_{\boldsymbol{x}^* \in \mathcal{X}(B,\underline{B},s)} \left\{ \|\boldsymbol{x}^{(k)}(\boldsymbol{x}^*) - \boldsymbol{x}^*\|_1 \right\}, \quad \gamma^{(k)} = 1, & \forall k \\ \beta^{(k)} \to& \mu(s-1), \quad p^{(k)} \to n, & \text{as } k \to \infty \end{aligned} \tag{17}$$

Despite remaining linearly convergent, the initial rate of our model $2\mu s - \mu$ is no worse than the rates in state-of-the-arts [7, 14, 1] and the eventual rate $1 - \sqrt{1 - \mu s + \mu}$ is strictly better than those models above due to the momentum. Under Assumption 1 in this paper, the eventual rate of LISTA-CPSS [7] is $\mu s - \mu$ and we have $\mu s - \mu > 1 - \sqrt{1 - \mu s + \mu}$ since $2\mu s - \mu < 1$. For example, if $\mu s - \mu = 0.9$, then our new rate is $1 - \sqrt{1 - \mu s + \mu} \approx 0.684$. Proof details can be found in the supplement.

3.3 Instance-Adaptive Parameters: From Linear to Superlinear Convergence

State-of-the-art theoretical results show that LISTA converges with a linear rate [7, 14, 1, 25, 28]. In ALISTA [14], the authors show that linear convergence is a lower bound. This means that one cannot expect a superlinear rate if a uniform set of optimal parameters are learned for a dataset.

One promise to enhance the rate in Theorem 1 is to further introduce instance adaptivity. We establish that, if $\{\theta^{(k)}, p^{(k)}, \gamma^{(k)}, \beta^{(k)}\}_{k=1}^{\infty}$ can be searched for an instance, such instance-optimal parameters lead to superlinear convergence.

Theorem 2. *Let $\boldsymbol{x}^{(0)} = \mathbf{0}$ and $\{\boldsymbol{x}^{(k)}\}_{k=1}^{\infty}$ be generated by (13). If Assumption 1 holds and $s < (1 + 1/\mu)/2$, then there exists a sequence of parameters $\{\theta^{(k)}, p^{(k)}, \gamma^{(k)}, \beta^{(k)}\}_{k=1}^{\infty}$ for each instance $\boldsymbol{x}^* \in \mathcal{X}(B, \underline{B}, s)$, we have:*

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_2 \leq \sqrt{s} B \prod_{t=0}^{k} \bar{c}^{(t)}, \quad \forall k = 1, 2, \cdots, \tag{18}$$

where $\{\bar{c}^{(k)}\}$ satisfies

$$\bar{c}^{(k)} \to 0 \text{ as } k \to \infty \tag{19}$$

and it holds that

- *the parameter sequence follows (17) as $k \leq \log(\underline{B}/sB)/\log(2\mu s - \mu)$,*
- *the model (13) reduces to the conjugate gradient (CG) iteration on the following linear system as $k > \log(\underline{B}/sB)/\log(2\mu s - \mu)$:*

$$\boldsymbol{W}_S^T \left(\boldsymbol{A}_S \boldsymbol{x}_S^* - \boldsymbol{b} \right) = \mathbf{0}. \tag{20}$$

3.4 HyperLISTA: Reducing ALISTA to Tuning Three Hyperparameters

Adaptive Parameters Theorem 2 suggests to use parameters following (17) at the initial stage. Based on the rules in (17), we design adaptive parameters in the following form

$$\theta^{(k)} = c_1 \mu \left\| \boldsymbol{A}^+ (\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}) \right\|_1, \tag{21}$$

$$\gamma^{(k)} = 1 \tag{22}$$

$$\beta^{(k)} = c_2 \mu \, \|\boldsymbol{x}^{(k)}\|_0, \tag{23}$$

$$p^{(k)} = c_3 \min \left(\log \left(\frac{\|\boldsymbol{A}^+ \boldsymbol{b}\|_1}{\|\boldsymbol{A}^+ (\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})\|_1} \right), n \right) \tag{24}$$

where $c_1 > 0, c_2 > 0, c_3 > 0$ are hyper-parameters to tune.

The step size formula (22) stems from the conclusion (17) in Theorem 1. The threshold $\theta^{(k)}$ in (17) cannot be directly used due to its dependency on the ground truth $\boldsymbol{x}^*$. We use $\|\boldsymbol{A}^+(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})\|_1$ to estimate $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_1$ and get formula (21). Similar thresholding schemes are studied in [30].

The momentum rule (23) is obtained via using the support of $\boldsymbol{x}^{(k)}$ to approximate the number of nonzeros s.

The support selection rule (24) can be explained as follows. First we sort the magnitude of $|x_i^*|$ as Figure 2 shows. At the k-th layer, given $e^{(k)} = \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_\infty$, each element in $\boldsymbol{x}^{(k)}$ can be bounded as

$$|x_i^*| - e^{(k)} \leq |x_i^{(k)}| \leq |x_i^*| + e^{(k)}, \quad \forall i.$$

Such area is shown as the gray area in Figure 2. As we discussed in Section 2 following definition (6), the support selection is a scheme to select those elements that can be trusted as true support. The "trust region" in Figure 2 illustrates such entries. Intuitively, the trust region should be larger as the error $e^{(k)}$ gets smaller. We propose to use the logarithmic function of the error to estimate $p^{(k)}$.

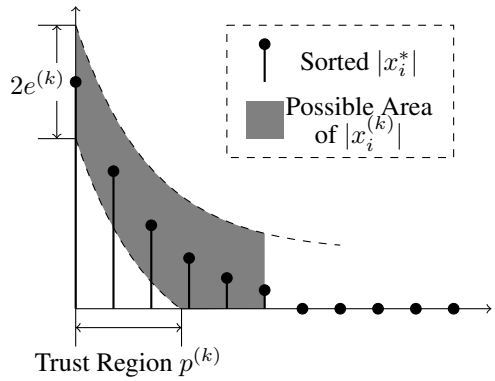


Figure 2: Choice of support selection

$$p^{(k)} \approx c_3 \log\left(\frac{\|\boldsymbol{x}^*\|_\infty}{e^{[}(k)}\right) \approx c_3 \log\left(\frac{\|\boldsymbol{A}^+\boldsymbol{b}\|_1}{\|\boldsymbol{A}^+(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})\|_1}\right) \tag{25}$$

The hyperparameter $0 < c_3 < 1$ is to tune based on the distribution of $\boldsymbol{x}^*$. With an upperbound n (the size of the vector $\boldsymbol{x}^*$), we obtain (24). Actually (25) can be mathematically derived by assuming the magnitude of the nonzero entries of $\boldsymbol{x}^*$ are normally distributed. (The derivation can be found in the supplement.) As $k \to \infty$, it holds that $e^{(k)} \to 0$ and $p^{(k)} \to n$, which means that the operator $\eta_{\theta^{(k)}}^{p^{(k)}}$ is getting close to a hard-thresholding operator. The support selection scheme proposed in LISTA-CPSS [7] also follows such principle but it requires more prior knowledge. Compared to that, formula (24) is much more automatic and self-adaptive.

Switching to Conjugate Gradients Theorem 2 suggests that one may call a conjugate gradient (CG) iteration to obtain faster convergence as the recovery signal $\boldsymbol{x}^{(k)}$ is accurate enough. In practice, we choose the switching condition as $p^{(k)}$ large enough. The linear system (20) to be solved by CG depends on the support of $\boldsymbol{x}^*$ and we estimate the support by the support of $\boldsymbol{x}^{(k)}$ since we assume $\boldsymbol{x}^{(k)}$ is accurate enough when we call CG. Finally, HyperLISTA is described in Algorithm 1.

Algorithm 1: HyperLISTA with tuned c_1, c_2, c_3

Input: Observed signal $\boldsymbol{b}$, hyperparameters c_1, c_2, c_3.
Initialize : Let $\boldsymbol{x}^{(0)} = \boldsymbol{0}$.
for $j = 0, 1, 2, \ldots$ *until convergence* **do**
  Conduct iteration (13) with parameters defined in (21 - 24).
  if $p^{(k)}$ *is large enough* **then**
    break
  end
end
Set $S = \mathrm{supp}(\boldsymbol{x}^{(k)})$.
Call the conjugate gradient algorithm to solve linear system (20).
Output: $\hat{\boldsymbol{x}}$, the result of the conjugate gradient.

3.5 Hyperparameter-Tuning Options for HyperLISTA

HyperLISTA has reduced the complexity of the learnable parameters to nearly the extreme: only three. While backpropagation remains to be a viable option to solve them, it becomes an over-kill to solve just three variables by passing gradients through tens of neural network layers. Besides, the hyperparameter c_3 in (23) decides the ratio of elements that will be selected into the support and hence passes the thresholding function; that makes c_3 non-differentiable in the computation graph.

Instead, we are allowed to seek simpler, even gradient-free methods to search for the parameters. We also empirically find HyperLISTA has certain robustness to perturbing the found values of c_1, c_2, c_3, which also encourages us to go with less precise yet much cheaper search methods.

In this work, we try replacing the backpropagation-based training with the vanilla grid search for learning HyperLISTA. Besides the ultra-light parameterization, another enabling factor is the low evaluation cost on fitness of hyperparameters: just running inference on one minibatch of training samples with them, and no training needed. Specifically, we we first use a coarse grid to find an "interested region" of hypermeters, and then zoom-in with a fine-grained grid. Details are to be found in Section 4.2, and we plan to try other options such as bayesian optimization [10] in future work.

4 Numerical experiments

Experiment settings We use synthesized datasets of 51,200 samples for training, 2,048 validation and 2,048 testing. We follow previous works [7, 14] to use a problem size of $(m, n) = (250, 500)$. The elements in the dictionary A are sampled from i.i.d. standard Gaussian distribution and we then normalize A to have unit ℓ_2 column norms. The sparse vectors x^* are sampled from $N(0, \sigma^2) \cdot Bern(p)$, where $N(\mu, \sigma^2)$ represents the normal distribution and $Bern(p)$ the Bernoulli distribution with probability p to take value 1. By default, we choose $\sigma = 1$ and $p = 0.1$, meaning that around 10% of the elements are non-zero whose magnitudes further follows a standard Gaussian distribution. The additive noise ε in (1) follows Gaussian distribution $N(0, \sigma_\varepsilon^2)$. The noise level is measured by signal-to-noise ratio (SNR) in decibel unit. Note that we can change the values of σ, p and σ_ε during testing to evaluate the adaptivity of models. In all experiments, the model performance is measured by normalized mean squared error (NMSE, same as defined in [7, 14]) in decibel unit. All models are unrolled and truncated to 16 layers for training and testing, unless otherwise specified.

4.1 Validation of symmetric matrix parameterization and Theorem 1

We first validate the efficacy of the symmetric matrix parameterization and Theorem 1, showing the acceleration brought by the momentum term. For this validation, we implement ALISTA and ALISTA with momentum, and also their variants with symmetric matrix parameterization. We then train them on the same dataset and compare their performance measured by NMSE on the testing set. Here, the support selection ratios are selected by following the recommendation in the official implementation of [14]. Results are shown in Figure 3[2]. As we can see from Figure 3, the symmetric matrix parameterization results in almost no performance drop, corroborating the validity of the new parameterization. On the other hand, introducing momentum to both parameterization brings significant improvement in NMSE.

4.2 Validation of instance-optimal parameter selection and grid search

In this subsection, we show that the adaptive parameterization in Eqns (21) - (24), i.e., HyperLISTA, can perform as well as or even better than the uniform parameterization in (13), and we also show the efficacy of the grid search method. We train HyperLISTA with backpropagation-based method (SGD) and grid search, respectively. The resulting models are denoted as HyperLISTA-BP and HyperLISTA-GS. As HyperLISTA-BP can not train c_3 in (24), we use the same manual support selection setting in ALISTA and train/search c_1 and c_2. Figure 4 show that grid search can find even better solutions than backpropagation and the uniform parameterization. When we further use grid search for searching c_1, c_2, c_3 simultaneously (HyperLISTA-Full), there is more performance boost.

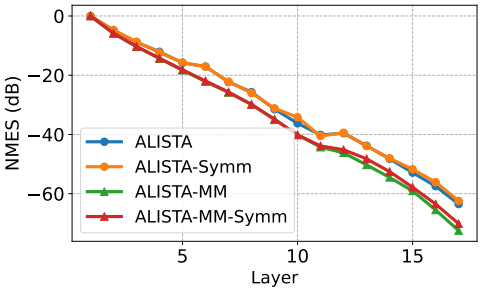


Figure 3: Effect of symmetric matrix parameterization and momentum. Momentum provides acceleration for either parameterization.

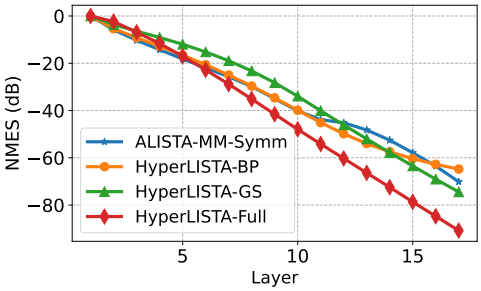
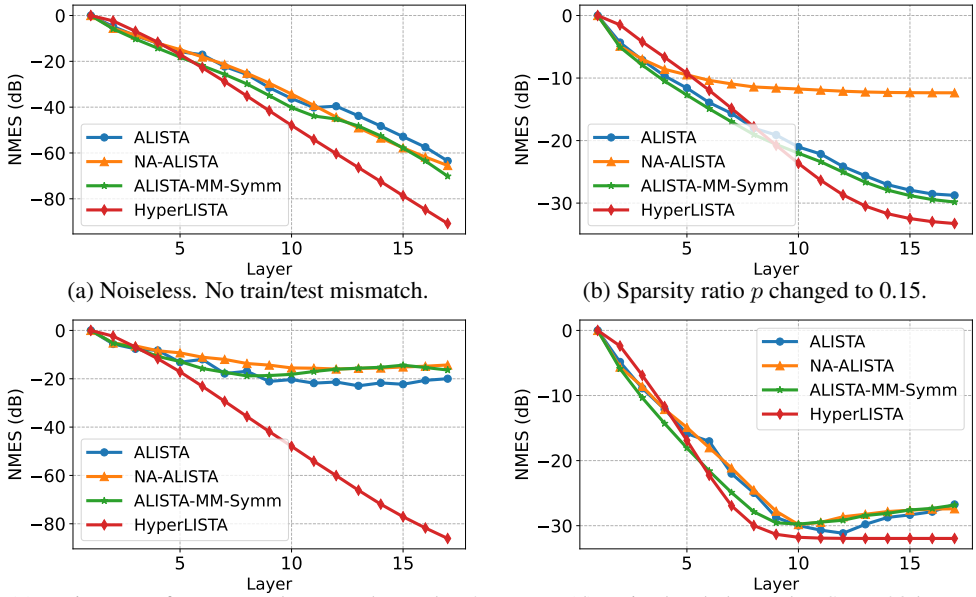


Figure 4: Back-propagation and grid search based training, compared with HyperLISTA-Full with c_1, c_2, c_3 searched.

[2]Here ALISTA achieves a worse NMSE than that in [14]. We attribute this to the finite-size training set that we use here instead of unlimited training set in [14].



(a) Noiseless. No train/test mismatch.

(b) Sparsity ratio p changed to 0.15.

(c) Variance σ of non-zero elements changed to 2.

(d) Noise level changed to SNR=30dB.

Figure 5: Adaptivity experiments. All models are trained in the noiseless case with $p = 0.1$, $\sigma = 1.0$, shown in subfigure (a). We directly apply the models trained in (a) to three different settings, which changes p to 0.15, σ to 2 and SNR of the noises to 30 respectively, shown in subfigures (b), (c), (d).

4.3 Adaptivity compared with previous work

We conduct experiments when there exists mismatch between the training and testing. We instantiate training/testing mismatch in terms of different sparsity levels in the sparse vectors x^*, different distributions of non-zero elements in x^*, and different additive noise levels. We compare HyperLISTA with ALISTA, and NA-ALISTA which also dynamically generates parameters per input using an external LSTM. Figure 5 show that in the original and all three mismatched cases, HyperLISTA achieves the best performance, especially showing excellent robustness to the non-zero magnitude distribution change (Fig. 5c). Note our computational overhead is also much lower than using LSTM.

Directly unrolling HyperLISTA to more layers Another hidden gem in HyperLISTA is that it learns iteration-independent hyperparameters, which, once searched and trained (on some fixed layer number), can be naturally extended to an arbitrary number of layers. Although NA-ALISTA can also be applied to any number of layers, it has worse performance and adaptivity compared to HyperLISTA, as we compare in Figure 6,

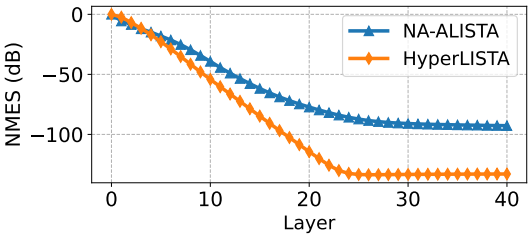


Figure 6: Direct unrolling to 40 layers (both HyperLISTA and NA-ALISTA are trained with 16 layers).

5 Conclusions and Discussions of Broad Impact

In this paper, we propose a ultra-light weight unrolling network, *HyperLISTA*, for solving sparse linear inverse problems. We first introduce a new ALISTA parameterization and augment it with momentum, and then propose an adaptive parameterization which reduces the training of HyperLISTA to only tuning three **instance- and layer-invariant** hyperparameters. HyperLISTA is theoretically proved and also empirically observed to have super-linear convergence rate as it uses instance-optimal parameters. Compared to previous work, HyperLISTA achieves faster convergence on the seen distribution (i.e., the training data distribution) and shows better adaptivity on unseen distributions. We do not see that this work will impose any social risks to the society. Besides the intellectual merits, the largest potential societal impact of this work is unrolling models that can be trained more efficiently and at the same time more are more adaptive, increasing the reusability.

References

[1] Aviad Aberdam, Alona Golts, and Michael Elad. Ada-lista: Learned solvers adaptive to varying models. *arXiv preprint arXiv:2001.08456*, 2020.

[2] Pierre Ablin, Thomas Moreau, Mathurin Massias, and Alexandre Gramfort. Learning step sizes for unfolded sparse coding. In *Advances in Neural Information Processing Systems*, pages 13100–13110, 2019.

[3] Pierre Ablin, Thomas Moreau, Mathurin Massias, and Alexandre Gramfort. Learning step sizes for unfolded sparse coding. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[4] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[5] Freya Behrens, Jonathan Sauder, and Peter Jung. Neurally augmented {alista}. In *International Conference on Learning Representations*, 2021.

[6] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, 2021.

[7] Xiaohan Chen, Jialin Liu, Zhangyang Wang, and Wotao Yin. Theoretical linear convergence of unfolded ista and its practical weights and thresholds. *arXiv preprint arXiv:1808.10038*, 2018.

[8] Benjamin Cowen, Apoorva Nandini Saridena, and Anna Choromanska. Lsalsa: accelerated source separation via learned sparse coding. *Machine Learning*, 108(8-9):1307–1327, 2019.

[9] Ingrid Daubechies, Michel Defrise, and Christine De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

[10] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham, 2019.

[11] Raja Giryes, Yonina C. Eldar, Alex M. Bronstein, and Guillermo Sapiro. Tradeoffs between convergence speed and reconstruction accuracy in inverse problems. *IEEE Transactions on Signal Processing*, 66(7):1676–1690, Apr 2018.

[12] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

[13] Satoshi Hara, Weichih Chen, Takashi Washio, Tetsuichi Wazawa, and Takeharu Nagai. Spod-net: Fast recovery of microscopic images using learned ista. In *Asian Conference on Machine Learning*, page 694–709, Oct 2019.

[14] Jialin Liu, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Alista: Analytic weights are as good as learned weights in lista. In *International Conference on Learning Representations (ICLR)*, 2019.

[15] Canyi Lu, Huan Li, and Zhouchen Lin. Optimized projections for compressed sensing via direct mutual coherence minimization. *Signal Processing*, 151:45–55, 2018.

[16] Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *arXiv preprint arXiv:1912.10557*, 2019.

[17] Thomas Moreau and Joan Bruna. Understanding trainable sparse coding via matrix factorization. *arXiv preprint arXiv:1609.00285*, 2016.

[18] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[19] Gregory Ongie, Ajil Jalal, Christopher A Metzler, Richard G Baraniuk, Alexandros G Dimakis, and Rebecca Willett. Deep learning techniques for inverse problems in imaging. *IEEE Journal on Selected Areas in Information Theory*, 1(1):39–56, 2020.

[20] Xi Peng, Ivor W Tsang, Joey Tianyi Zhou, and Hongyuan Zhu. k-meansnet: When k-means meets differentiable programming. *arXiv preprint arXiv:1808.07292*, 2018.

[21] Dimitris Perdios, Adrien Besson, Philippe Rossinelli, and Jean-Philippe Thiran. Learning the weight matrix for sparsity averaging in compressive imaging. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3056–3060. IEEE, 2017.

[22] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[23] Pablo Sprechmann, Roee Litman, Tal Ben Yakar, Alexander M Bronstein, and Guillermo Sapiro. Supervised sparse analysis and synthesis operators. *Advances in Neural Information Processing Systems*, 26:908–916, 2013.

[24] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[25] Kailun Wu, Yiwen Guo, Ziang Li, and Changshui Zhang. Sparse coding with gated learned ista. In *International Conference on Learning Representations*, 2020.

[26] Jinxi Xiang, Yonggui Dong, and Yunjie Yang. Fista-net: Learning a fast iterative shrinkage thresholding network for inverse problems in imaging. *IEEE Transactions on Medical Imaging*, 40(5):1329–1339, 2021.

[27] Bo Xin, Yizhou Wang, Wen Gao, David Wipf, and Baoyuan Wang. Maximal sparsity with deep networks? *Advances in Neural Information Processing Systems*, 29:4340–4348, 2016.

[28] Chengzhu Yang, Yuantao Gu, Badong Chen, Hongbing Ma, and Hing Cheung So. Learning proximal operator methods for nonconvex sparse recovery with theoretical guarantee. *IEEE Transactions on Signal Processing*, 68:5244–5259, 2020.

[29] Joey Tianyi Zhou, Kai Di, Jiawei Du, Xi Peng, Hao Yang, Sinno Jialin Pan, Ivor W Tsang, Yong Liu, Zheng Qin, and Rick Siow Mong Goh. Sc2net: Sparse lstms for sparse coding. In *AAAI*, pages 4588–4595, 2018.

[30] Li Ziang, Wu Kailun, Yiwen Guo, and Changshui Zhang. Learned ista with error-based thresholding for adaptive sparse coding, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [No]
   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not see that this work will impose any social risks to the society.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We will publish them upon acceptance.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] We are using synthesized data.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We are using synthesized data.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We are using synthesized data.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]